PORTAL
US Patent & Trademark Office

Subscribe (Full Service) Register (Limited Service, Free) Login

Search: ◉ The ACM Digital Library ○ The Guide

loop invariant range SEARCH

THE ACM DIGITAL LIBRARY

Feedback Report a problem Satisfaction survey

Terms used **loop invariant range** Found **17,381** of **132,857**

Sort results by: relevance
Display results: expanded form
Save results to a Binder
Search Tips
☐ Open results in a new window

Try an Advanced Search
Try this search in The ACM Guide

Results 1 - 20 of 200 Result page: **1** 2 3 4 5 6 7 8 9 10 next
Best 200 shown Relevance scale ■■■■■

1 Optimization of array subscript range checks
Jonathan M. Asuru
June 1992 **ACM Letters on Programming Languages and Systems (LOPLAS)**, Volume 1 Issue 2

Full text available: pdf(619.54 KB) Additional Information: full citation, abstract, references, citings, index terms, review

Compile-time elimination of subscript range checks is performed by some optimizing compilers to reduce the overhead associated with manipulating array data structures. Elimination and propagation, the two methods of subscript range check optimization, are less effective for eliminating global redundancies especially in while-loop structures with nonconstant loop guards. This paper describes a subscript range check optimization procedure that can eliminate more range checks than current meth ...

Keywords: conservative expression substitution, loop guard elimination, range check optimization

2 Session 18: dependence analysis/loop parallelization: The range test: a dependence test for symbolic, non-linear expressions
William Blume, Rudolf Eigenmann
November 1994 **Proceedings of the 1994 ACM/IEEE conference on Supercomputing**

Full text available: pdf(914.29 KB) Additional Information: full citation, abstract, references

Most current data dependence tests cannot handle loop bounds or array subscripts that are symbolic, nonlinear expressions (e.g. A (n*i+j), where $0 \leqslant j \leqslant n$). In this paper, we describe a dependence test, called the range test, that can handle such expressions. Briefly, the range test proves independence by determining whether certain symbolic inequalities hold for a permutation of the loop nest. Powerful symbolic analyses and constraint propagation techniques were developed to prove such in ...

3 Elimination of redundant array subscript range checks
Priyadarshan Kolte, Michael Wolfe
June 1995 **ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 1995 conference on Programming language design and implementation**, Volume 30 Issue 6

Full text available: pdf(1.09 MB) Additional Information: full citation, abstract, references, citings, index terms

This paper presents a compiler optimization algorithm to reduce the run time overhead of

array subscript range checks in programs without compromising safety. The algorithm is based on partial redundancy elimination and it incorporates previously developed algorithms for range check optimization. We implemented the algorithm in our research compiler, Nascent, and conducted experiments on a suite of 10 benchmark programs to obtain four results: (1) the execution overhead of naive range check ...

4 Programming pearls: Writing correct programs
Jon Bentley
December 1983 **Communications of the ACM**, Volume 26 Issue 12

Full text available: pdf(648.55 KB) Additional Information: full citation, abstract

In the late 1960s people were talking about the promise of programs that verify the correctness of other programs. Unfortunately, it is now the middle of the 1980s, and, with precious few exceptions, there is still little more than talk about automated verification systems. Despite unrealized expectations, however, the research on program verification has given us something far more valuable than a black box that gobbles programs and flashes "good" or "bad"—we ...

5 A reexamination of "Optimization of array subscript range checks"
Wei-Ngan Chin, Eak-Khoon Goh
March 1995 **ACM Transactions on Programming Languages and Systems (TOPLAS)**, Volume 17 Issue 2

Full text available: pdf(638.50 KB) Additional Information: full citation, abstract, references, citings, index terms, review

Jonathan Asuru proposed recently an enhanced method for optimizing array subscript range checks. The proposed method is however unsafe and may generate optimized programs whose behavior is different from the original program. Two main flaws in Asuru's method are described, together with suggested remedies and improvements.

Keywords: backward checks propagation, integer programming, loop guard elimination, safe bound checks optimization

6 Application of the goal invariant to the structuring of programs
Janusz W. Laski
January 1982 **ACM SIGPLAN Notices**, Volume 17 Issue 1

Full text available: pdf(774.02 KB) Additional Information: full citation, abstract, references

Functional rather than structural approach to conversion of a two-exit loop into a <u>while</u> loop is presented. The functions computed by the corresponding exits are formally specified and their union is analyzed to reveal the cause of its loop nonrealizability. The latter is removed by either redefining the domain/range of the constituent functions or by inclusion of the loop postludes into the loop specification. None of these methods changes the functionality of the original pr ...

Keywords: Structured programs, domain tuning, double exit loop, function, goal invariant, loop realizability, postludes, range tuning, termination assertion

7 Practical data breakpoints: design and implementation
Robert Wahbe, Steven Lucco, Susan L. Graham
June 1993 **ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 1993 conference on Programming language design and implementation**, Volume 28 Issue 6

Full text available: pdf(1.37 MB) Additional Information: full citation, abstract, references, citings, index terms

A data breakpoint associates debugging actions with programmer-specified conditions on the memory state of an executing program. Data breakpoints provide a means for discovering program bugs that are tedious or impossible to isolate using control breakpoints alone. In practice, programmers rarely use data breakpoints, because they are either unimplemented or prohibitively slow in available debugging software. In this paper, we present the design and implementation of a practical data breakp ...

8 Heuristics for program synthesis using loop invariants
Joe W. Duran
January 1978 **Proceedings of the 1978 annual conference - Volume 2**
Full text available: pdf(535.95 KB) Additional Information: full citation, abstract, references, index terms

The problem of automatically synthesizing programs from formal specifications can be stated as, "constructively prove that there exists a program P which transforms valid input into output satisfying the required output assertion." If Requires a loop, the proof will contain an induction. This is beyond the ability of current theorem provers and synthesis systems. In this paper, the loop invariant is used to reduce the problem of synthesizing loops to that of synthesizing three ...

Keywords: Automatic programming, Loop invariants, Program correctness, Program synthesis

9 Fusion-based register allocation
Guei-Yuan Lueh, Thomas Gross, Ali-Reza Adl-Tabatabai
May 2000 **ACM Transactions on Programming Languages and Systems (TOPLAS)**, Volume 22 Issue 3
Full text available: pdf(475.45 KB) Additional Information: full citation, abstract, references, citings, index terms, review

The register allocation phase of a compiler maps live ranges of a program to registers. If there are more candidates than there are physical registers, the register allocator must spill a live range (the home location is in memory) or split a live range (the live range occupies multiple locations). One of the challenges for a register allocator is to deal with spilling and splitting together. Fusion-based register allocation uses the structure of the program to make splitting and spilling d ...

Keywords: performance evaluation, register allocation

10 Practical program verification: automatic program proving for real-time embedded software
John Nagle, Scott Johnson
January 1983 **Proceedings of the 10th ACM SIGACT-SIGPLAN symposium on Principles of programming languages**
Full text available: pdf(739.44 KB) Additional Information: full citation, abstract, references

Despite the attractiveness of the concept, attempts to date to use proof of correctness techniques on production software have been generally unsuccessful. The obstacles encountered are not fundamental. We have implemented a proof of correctness system to be used for improving the realiability of certain small, real-time programs. It appears that many of the problems of past systems can be avoided.This work is supported by the Long Range Research Program of the Ford Motor Company, Dearborn, Mich ...

11 A fresh look at optimizing array bound checking
Rajiv Gupta
June 1990 **ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 1990 conference**

on Programming language design and implementation, Volume 25 Issue 6

Full text available: pdf(998.53 KB) Additional Information: full citation, abstract, references, citings, index terms

This paper describes techniques for optimizing range checks performed to detect array bound violations. In addition to the elimination of range checks, the optimizations discussed in this paper also reduce the overhead due to range checks that cannot be eliminated by compile-time analysis. The optimizations reduce the program execution time and the object code size through elimination of redundant checks, propagation of checks out of loops, and combination of multiple checks into a single c ...

12 Subsequence references: first-class values for substrings

Wilfred J. Hansen

October 1992 **ACM Transactions on Programming Languages and Systems (TOPLAS)**, Volume 14 Issue 4

Full text available: pdf(1.31 MB) Additional Information: full citation, abstract, references, citings, index terms, review

Arrays of characters are a basic data type in many programming languages, but strings and substrings are seldom accorded first-class status as parameters and return values. Such status would enable a routine that calls a search function to readily access context on both sides of a return value. To enfranchise substrings, this paper describes a new data type for substrings as a special case of one for general subsequences. The key idea is that values are not sequences or references to positi ...

Keywords: ATK, AUIS, Andrew Toolkit, Ness, document processing, programming language design, sequences, string searching, strings, subsequences, substrings

13 Dynamically discovering likely program invariants to support program evolution

Michael D. Ernst, Jake Cockrell, William G. Griswold, David Notkin

May 1999 **Proceedings of the 21st international conference on Software engineering**

Full text available: pdf(1.59 MB) Additional Information: full citation, references, citings, index terms

Keywords: dynamic analysis, execution traces, formal specification, logical inference, pattern recognition, program invariants, software evolution

14 On defining application-specific high-level array operations by means of shape-invariant programming facilities

Sven-Bodo Scholz

July 1998 **ACM SIGAPL APL Quote Quad , Proceedings of the APL98 conference on Array processing language**, Volume 29 Issue 3

Full text available: pdf(583.03 KB) Additional Information: full citation, abstract, references, citings, index terms

Most of the existing high-level array-processing languages support a fixed set of pre-defined array *operations* and a few higher-order functions for constructing new array operations from existing ones. In this paper, we discuss a more general approach made feasible by SAC (for Single Assignement C), a functional variant of C.SAC provides a meta-level language construct called WITH-loop which may be considered a sophisticated variant of the FORALL-loops ...

Keywords: compilation, high-level array operations, meta-level programming, performance comparison, shape-invariant programming

15 Structured Programming with go to Statements
Donald E. Knuth
December 1974 **ACM Computing Surveys (CSUR)**, Volume 6 Issue 4
Full text available: pdf(3.02 MB) Additional Information: full citation, references, citings, index terms

16 Debugging highly-optimized Ada with code motion (DHACM)
Kevin Tucker
November 1997 **Proceedings of the conference on TRI-Ada '97**
Full text available: pdf(721.40 KB) Additional Information: full citation, references, index terms

17 Optimization of expressions in Fortran
Vincent A. Busam, Donald E. Englund
December 1969 **Communications of the ACM**, Volume 12 Issue 12
Full text available: pdf(1.19 MB) Additional Information: full citation, abstract, references, citings, index terms

A method of optimizing the computation of arithmetic and indexing expressions of a Fortran program is presented. The method is based on a linear analysis of the definition points of the variables and the branching and DO loop structure of the program. The objectives of the processing are (1) to eliminate redundant calculations when references are made to common sub-expression values, (2) to remove invariant calculations from DO loops, (3) to efficiently compute subscripts contain ...

Keywords: DO loops, FORTRAN, common subexpressions, compilation, compilers, expressions, invariant calculations, optimization, register allocation, subscripts

18 Effective sign extension elimination
Motohiro Kawahito, Hideaki Komatsu, Toshio Nakatani
May 2002 **ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 2002 Conference on Programming language design and implementation**, Volume 37 Issue 5
Full text available: pdf(389.09 KB) Additional Information: full citation, abstract, references, index terms

Computer designs are shifting from 32-bit architectures to 64-bit architectures, while most of the programs available today are still designed for 32-bit architectures. Java™, for example, specifies the frequently used int" as a 32-bit data type. If such Java programs are executed on a 64-bit architecture, many 32-bit values must be sign-extended to 64-bit values for integer operations. This causes serious performance overhead. In this paper, we present a fast and effective algorithm for e ...

Keywords: 64-bit architectures, IA-64, JIT compilers, Java, sign extension

19 Automatic program specialization for Java
Ulrik P. Schultz, Julia L. Lawall, Charles Consel
July 2003 **ACM Transactions on Programming Languages and Systems (TOPLAS)**, Volume 25 Issue 4
Full text available: pdf(1.18 MB) Additional Information: full citation, abstract, references, citings, index terms

The object-oriented style of programming facilitates program adaptation and enhances

program genericness, but at the expense of efficiency. We demonstrate experimentally that state-of-the-art Java compilers fail to compensate for the use of object-oriented abstractions in the implementation of generic programs, and that program specialization can eliminate a significant portion of these overheads. We present an automatic program specializer for Java, illustrate its use through detailed case stud ...

Keywords: Automatic program specialization, Java, object-oriented languages, optimization, partial evaluation

20 Program optimization and parallelization using idioms

Shlomit S. Pinter, Ron Y. Pinter

January 1991 **Proceedings of the 18th ACM SIGPLAN-SIGACT symposium on Principles of programming languages**

Full text available: pdf(1.23 MB) Additional Information: full citation, references, citings, index terms

Results 1 - 20 of 200 Result page: **1** 2 3 4 5 6 7 8 9 10 next

The ACM Portal is published by the Association for Computing Machinery. Copyright © 2004 ACM, Inc.
Terms of Usage Privacy Policy Code of Ethics Contact Us

Useful downloads: Adobe Acrobat QuickTime Windows Media Player Real Player

Freeform Search

Database: US Pre-Grant Publication Full-Text Database
US Patents Full-Text Database
US OCR Full-Text Database
EPO Abstracts Database
JPO Abstracts Database
Derwent World Patents Index
IBM Technical Disclosure Bulletins

Term: loop same invariant same range

Display: 50 **Documents in Display Format:** REV **Starting with Number** 1

Generate: ○ **Hit List** ◉ **Hit Count** ○ **Side by Side** ○ **Image**

Search Clear Interrupt

Search History

DATE: Wednesday, May 12, 2004 Printable Copy Create Case

Set Name side by side	Query	Hit Count	Set Name result set
DB=USPT; PLUR=NO; OP=OR			
L5	loop same invariant same range	55	L5
L4	loop same invariant same bound same range	0	L4
L3	L2 AND (upper ADJ bound)	112	L3
L2	L1 ANd range	948	L2
L1	Loop AND invariant ANd bound	1091	L1

END OF SEARCH HISTORY

Refine Search

Search Results -

Terms	Documents
5787286.pn. or 5644709.pn. 5515081.pn. 6526421.pn. or 5696974.pn. or 5819088.pn. or 5835776.pn. 5819088.pn. or 6163838.pn.	8

Database:
US Pre-Grant Publication Full-Text Database
US Patents Full-Text Database
US OCR Full-Text Database
EPO Abstracts Database
JPO Abstracts Database
Derwent World Patents Index
IBM Technical Disclosure Bulletins

Search: L1

Refine Search

Recall Text | Clear | Interrupt

Search History

DATE: Wednesday, May 12, 2004 Printable Copy Create Case

Set Name side by side	Query	Hit Count	Set Name result set
	DB=USPT; PLUR=NO; OP=OR		
L1	5787286.pn. or 5644709.pn. 5515081.pn. 6526421.pn. or 5696974.pn. or 5819088.pn. or 5835776.pn. 5819088.pn. or 6163838.pn.	8	L1

END OF SEARCH HISTORY

Refine Search

Search Results -

Terms	Documents
L5 AND range	14

Database:
US Pre-Grant Publication Full-Text Database
US Patents Full-Text Database
US OCR Full-Text Database
EPO Abstracts Database
JPO Abstracts Database
Derwent World Patents Index
IBM Technical Disclosure Bulletins

Search: L6

Refine Search | Recall Text | Clear | Interrupt

Search History

DATE: Wednesday, May 12, 2004 Printable Copy Create Case

Set Name side by side	Query	Hit Count	Set Name result set
	DB=USPT; PLUR=NO; OP=OR		
L6	L5 AND range	14	L6
L5	L4 AND loop and invariant	25	L5
L4	(5276819 5390325 5490249 5652835 6151706 4782444 4991088 5226128 5297291 5361354 5440723 5448737 5452442 5481708 5485575 5511198 5535391 5625835 5787284 5850549 5854934 5862384 5892854 6128775 6145120 6286135 6625737 6219438 6219438 5485600 6237135 6434447 6003024 6233573 4782284 4811206 4931941 4952862 4990840 5182656 5249259 5276828 5286962 5354961 5542040 5722427 5842022 5850348 5908464 5936383).pn.	49	L4
L3	6457076.pn.	1	L3
L2	6151643.pn.	1	L2
L1	6151643	51	L1

END OF SEARCH HISTORY